

# TP FreeRTOS

L'objectif de ce TP est de mettre en place quelques applications sous FreeRTOS en utilisant la carte NUCLEO-G431RB conçue autour du STM32G431RBT6.

Tout au long du TP, la fonction `printf()` sera beaucoup utilisée. Pour rendre celà possible, il faut ajouter les lignes suivantes, dans le fichier `main.c`, avant la fonction `main()` :

```
int __io_putchar(int ch) {  
    HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, HAL_MAX_DELAY);  
    return ch;  
}
```

## 1 FreeRTOS, tâches et sémaphores

### 1.1 Tâche simple

1. Vous pouvez travailler dans le projet créé à la partie précédente Activez FreeRTOS et notez les paramètres qui vous paraissent pertinents. En quoi le paramètre `TOTAL_HEAP_SIZE` a-t-il de l'importance ?  
Observez l'impact de votre configuration sur le fichier `FreeRTOSConfig.h`
2. Créez une tâche permettant de faire changer l'état de la LED toutes les 100ms et profitez-en pour afficher du texte à chaque changement d'état. Quel est le rôle de la macro `portTICK_PERIOD_MS` ?

### 1.2 Sémaphores pour la synchronisation

3. Créez deux tâches, **taskGive** et **taskTake**, ayant deux priorités différentes. **TaskGive** donne un sémaphore toutes les 100ms. Affichez du texte avant et après avoir donné le sémaphore. **TaskTake** prend le sémaphore. Affichez du texte avant et après avoir pris le sémaphore.
4. Ajoutez un mécanisme de gestion d'erreur lors de l'acquisition du sémaphore. On pourra par exemple invoquer un *reset software* au STM32 si le sémaphore n'est pas acquis au bout d'une seconde.
5. Pour valider la gestion d'erreur, ajoutez 100ms au délai de **TaskGive** à chaque itération.
6. Changez les priorités. Expliquez les changements dans l'affichage.

### 1.3 Notification

7. Modifiez le code pour obtenir le même fonctionnement en utilisant des *task notifications* à la place des sémaphores.

## 1.4 Queues

8. Modifiez **TaskGive** pour envoyer dans une *queue* la valeur du timer. Modifiez **TaskTake** pour réceptionner et afficher cette valeur.

## 1.5 Réentrance et exclusion mutuelle

```
#define STACK_SIZE 256
```

```
#define TASK1_PRIORITY 1
```

```
#define TASK2_PRIORITY 2
```

```
#define TASK1_DELAY 1
```

```
#define TASK2_DELAY 2
```

```
ret = xTaskCreate(task_bug, "Tache 1", STACK_SIZE, \  
    (void *) TASK1_DELAY, TASK1_PRIORITY, NULL);
```

```
configASSERT(pdPASS == ret);
```

```
ret = xTaskCreate(task_bug, "Tache 2", STACK_SIZE, \  
    (void *) TASK2_DELAY, TASK2_PRIORITY, NULL);
```

```
configASSERT(pdPASS == ret);
```

```
void task_bug(void * pvParameters)
```

```
{
```

```
    int delay = (int) pvParameters;
```

```
    for(;;)
```

```
    {
```

```
        printf("Je suis %s et je m'endors pour \  
                %d ticks\r\n", pcTaskGetName(NULL), delay);
```

```
        vTaskDelay(delay);
```

```
    }
```

```
}
```

9. Recopiez le code ci-dessus – au bon endroit – dans votre code.
10. Observez attentivement la sortie dans la console. Expliquez d'où vient le problème.
11. Proposez une solution en utilisant un sémaphore Mutex.

## 2 On joue avec le Shell

### Attention !

Pour ce TP, il y a une petite subtilité. Seules les interruptions dont la priorité est supérieure à la valeur `configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY` (définie à 5 par défaut) peuvent appeler des primitives de FreeRTOS. On peut soit modifier ce seuil, soit modifier la priorité de l'interruption de l'USART1 (0 par défaut). Dans l'exemple montré en Figure 1, la priorité de l'interruption de l'USART1 est fixée à 5.

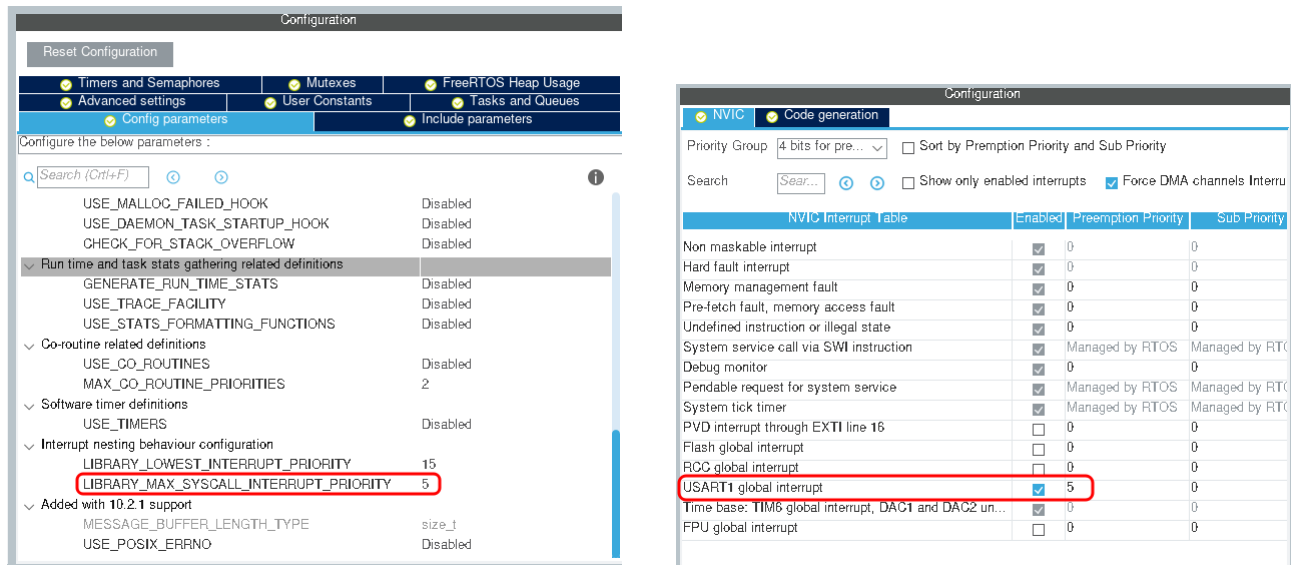


Figure 1 – Priorité maximale définie dans FreeRTOS (à gauche) et priorité à définir pour l'USART1 dans le NVIC (à droite)

1. Terminer l'intégration du shell commencé en TD. Pour mémoire, les questions du TD sont rappelées ci-dessous :
  1. Créer le projet, compiler et observer. Appeler la fonction depuis le shell. Les fichiers sont disponibles sur moodle, dans la section TD.
  2. Modifier la fonction pour faire apparaître la liste des arguments.
  3. Expliquer les mécanismes qui mènent à l'exécution de la fonction.
  4. Quel est le problème ?
  5. Proposer une solution
2. Que se passe-t-il si l'on ne respecte pas les priorités décrites précédemment ?
3. Écrire une fonction `led()`, callable depuis le shell, permettant de faire clignoter la LED (PI1 sur la carte). Un paramètre de cette fonction configure la période de clignotement. Une valeur de 0 maintient la LED éteinte.
 

Le clignotement de la LED s'effectue dans une tâche. Il faut donc trouver un moyen de faire communiquer **\*proprement\*** la fonction `led` avec la tâche de clignotement.

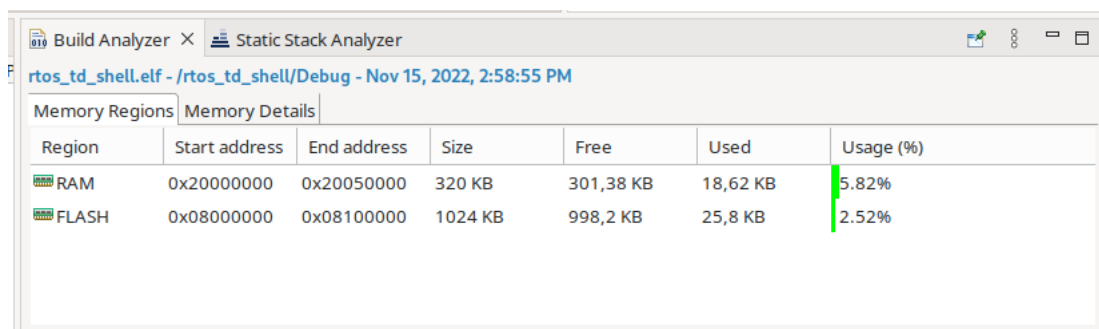
### 3 Debug, gestion d'erreur et statistiques

Ce TP se réalise dans le même projet, à la suite du TP précédent. On part donc du principe que le shell est fonctionnel et utilise un mécanisme d'OS (sémaphore, queue ou notification) pour la synchronisation avec une interruption.

#### 3.1 Gestion du tas

Un certain nombre de fonctions de l'OS peuvent échouer. Les fonctions finissant par `Create` font de l'allocation dynamique et peuvent échouer s'il n'y a plus assez de mémoire.

1. Quel est le nom de la zone réservée à l'allocation dynamique ?
2. Est-ce géré par FreeRTOS ou la HAL ?
3. Si ce n'est déjà fait, ajoutez de la gestion d'erreur sur toutes les fonctions pouvant générer des erreurs. En cas d'erreur, affichez un message et appelez la fonction `Error_Handler()` ;
4. Notez la mémoire RAM et Flash utilisée, comme dans l'exemple ci-dessous



The screenshot shows the 'Static Stack Analyzer' window with the 'Memory Details' tab selected. It displays a table of memory regions for the file 'rtos\_td\_shell.elf'.

Region	Start address	End address	Size	Free	Used	Usage (%)
RAM	0x20000000	0x20050000	320 KB	301,38 KB	18,62 KB	5.82%
FLASH	0x08000000	0x08100000	1024 KB	998,2 KB	25,8 KB	2.52%

5. Créez des tâches bidons jusqu'à avoir une erreur.
6. Notez la nouvelle utilisation mémoire. Pourquoi n'a-t-elle pas changé ?
7. Dans CubeMX, augmentez la taille du tas (`TOTAL_HEAP_SIZE`). Générez le code, compilez et testez.
8. Notez la nouvelle utilisation mémoire. Expliquez les trois relevés.

#### 3.2 Gestion des piles

Dans cette partie du TP, vous allez utiliser un *hook* (une fonction appelée par l'OS, dont on peut écrire le contenu) pour détecter les dépassements de pile (*Stack Overflow* en anglais).

1. Lisez la doc suivante :  
<https://www.freertos.org/Stacks-and-stack-overflow-checking.html>
2. Dans CubeMX, configurez `CHECK_FOR_STACK_OVERFLOW`
3. Écrivez la fonction `vApplicationStackOverflowHook`. (Rappel : C'est une fonction appelée automatiquement par FreeRTOS, vous n'avez pas à l'appeler vous-même).

4. Débrouillez vous pour remplir la pile d'une tâche pour tester. Notez que, vu le contexte d'erreur, il ne sera peut-être pas possible de faire grand chose dans cette fonction. Utilisez le debugger.
5. Il existe d'autres *hooks*. Expliquez l'intérêt de chacun d'entre eux.

### 3.3 Statistiques dans l'IDE

On peut afficher un certain nombre d'informations relatives à FreeRTOS dans STM32CubeIDE en mode debug.

1. Dans CubeMX, activez les trois paramètres suivants :
  - GENERATE\_RUN\_TIME\_STATS
  - USE\_TRACE\_FACILITY
  - USE\_STATS\_FORMATTING\_FUNCTIONS
2. Générez le code, compilez et lancez en mode debug
3. Pour ajouter les statistiques, cliquez sur Window > Show View > FreeRTOS > FreeRTOS Task List. Vous pouvez aussi afficher les queues et les sémaphores.
4. Lancez le programme puis mettez-le en pause pour voir les statistiques.
5. Cherchez dans CubeMX comment faire pour afficher l'utilisation de la pile. En mode debug, cliquez sur Toggle Stack Checking (dans l'onglet FreeRTOS Task List en haut à droite).
6. Pour afficher le taux d'utilisation du CPU, il faut écrire les deux fonctions suivantes :

```
void configureTimerForRunTimeStats(void);  
unsigned long getRunTimeCounterValue(void);
```

La première fonction doit démarrer un timer, la seconde permet de récupérer la valeur du timer. Si vous utilisez un timer 16 bits, il faudra peut-être bricoler un peu.

Encore une fois, ce sont des *hooks*, elles sont donc automatiquement appelées par l'OS.

7. Affichez les sémaphores et les queues.
8. Si vous n'en utilisez pas dans votre projet, créez deux tâches qui se partagent une queue ou un sémaphore.
9. Pour leur donner un nom compréhensible, utilisez la fonction `vQueueAddToRegistry`.

### 3.4 Affichage des statistiques dans le shell

Vous pouvez vous référer à la documentation de FreeRTOS en suivant ce lien : <https://www.freertos.org/rtos-run-time-stats.html>. Deux fonctions seront utiles à cette partie du TP :

```
void vTaskGetRunTimeStats(char * pcWriteBuffer);  
void vTaskList(char * pcWriteBuffer);
```

1. Écrire une fonction callable depuis le shell pour afficher les statistiques dans le terminal.